```
// satori.cpp
// Copyright (c) 2003. Satori Labs, Inc. All Rights Reserved.
// Note that in the code examples below, three dots (. . .) indicates places
// where code that wasn't relevant to illustrating the point was removed.
// Understanding this code requires knowledge of C++, Microsoft Pocket PC
// programming, some MFC, and some familiarity with Anoto pen technology.
//*************************************************************************
// 1) Receiving notification from the pen:
// Our current system receives a file from the pen via a Bluetooth "push";
// we set up to be notified when a new file appears in the
// appropriate directory
void CFusionWareEditorView::OnInitialUpdate()
{
   // . . .
   // ask to be notified of a file change in the PGD target directory, which may be a pgd file.
   // IFF widcomm exists, use it's path set for  incoming files, else
   // default to temp dir.
   TCHAR buf[MAX_PATH];
   _tcscpy(buf, _T("\\Temp"));
   LPCTSTR lpszWidcommInboxDir = _T("SOFTWARE\\WIDCOMM\\btConfig\\Services\\0002");
   CRegKey cKey;
   LONG lResult = cKey.Open(HKEY_LOCAL_MACHINE, lpszWidcommInboxDir);
   if (ERROR_SUCCESS == lResult)
   {
      DWORD dwCount = MAX_PATH;
      cKey.QueryValue(buf, _T("InboxDirectory"), &dwCount);
      cKey.Close();
   }
   if (_tcslen(buf))
   {
      SHCHANGENOTIFYENTRY cne;
      cne.dwEventMask = SHCNE_CREATE;
      cne.pszWatchDir = buf;
      cne.fRecursive = FALSE;
      if (!SHChangeNotifyRegister(GetSafeHwnd(), &cne))
      {
         this->MessageBox(_T("Failed to register for pgd change notification!"), g_strAppName);
      }
   }
   // . . .
}
// We then wait for a change notification, and try to handle it.
// The parser determines if its a valid pgd file.
// The file format, and as a result the parser, are defined by Anoto,
// the pen manufacturer.
LRESULT CFusionWareEditorView::OnFileChangeInfo(WPARAM /* wParam */, LPARAM lParam)
{
   FILECHANGENOTIFY *lpfcn = (FILECHANGENOTIFY*)lParam;
   FILECHANGEINFO *lpfci;
   lpfci = &(lpfcn->fci);
```

```
    ASSERT(lpfci->wEventId == SHCNE_CREATE);
    // this calls code to actually parse the data
    GetDocument()->HandleNewPGD((LPCTSTR)lpfci->dwItem1);
    SHChangeNotifyFree(lpfcn);
    return TRUE;
}
// ****************************************************************
// 2)  Displaying handwriting on pgd screen
//
// We add the strokes for a given page here
void CEditItemDlg::SetItemInfo( void* pCurrentItem, CPtrArray* pItemsArray, int iCurrentIndex )
{
    for (int i = 0; i < page->m_drawAreas->GetSize(); i++)
    {
        PadDrawArea &pdArea = page->m_drawAreas->ElementAt(i);
        int nSize = pdArea.m_userAreas->GetSize();
        for (int j = 0; j < nSize; j++)
        {
            PadArea &pArea = pdArea.m_userAreas->ElementAt(j);
            if (pArea.m_strokes)
            {
                int numStrokes = pArea.m_strokes->GetSize();
                // add each stroke - I will probably change this to just pass in the m_strokes list itself
                // The current one stroke at a time is based on an old algorithm
                for (int i = 0; i < numStrokes; i++)
                    m_graphicView.AddStroke(pArea.m_strokes->ElementAt(i));
            }
        }
    }
}
void CBitmapDisplay::AddStroke(PenStroke &newStroke)
{
    m_strokes.Add(newStroke);
    m_bRecreate = TRUE;
    m_bShowHorz = m_bShowVert = FALSE;
}
// everything is placed in a memory device context, so paint just blits that to the screen
void CBitmapDisplay::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    if (m_bRecreate)
        CreateMemDC();
    if (m_memDC)
        dc.BitBlt(m_nOffsetx, m_nOffsety, m_sizeClient.cx, m_sizeClient.cy, m_memDC, m_nSourcex, m_nSourcey,
SRCCOPY);
    m_erase = FALSE;
}
// here is where we draw the PAD file contents, and the the pen strokes to create a view of the page.
BOOL CBitmapDisplay::CreateMemDC()
{
```

```
if (!m_bRecreate || (!m_bmpNew.m_hObject && !m_padPage))
    return FALSE;
if (m_memDC)
    delete m_memDC;
CRect cRect;
GetClientRect(cRect);
m_sizeClient.cx = cRect.Width() - 18;
m_sizeClient.cy = cRect.Height() - 18;
if (!m_padPage)
{
    m_sizeAll.cx = (m_bmInfo.bmWidth > m_sizeClient.cx) ? m_bmInfo.bmWidth : m_sizeClient.cx;
    m_sizeAll.cy = (m_bmInfo.bmHeight > m_sizeClient.cy) ? m_bmInfo.bmHeight : m_sizeClient.cy;
    m_memDC = new CDC();
    CClientDC dc(this);
    m_memDC->CreateCompatibleDC(&dc);
    CDC tmpDC;
    tmpDC.CreateCompatibleDC(&dc);
    m_bmpOld = (HBITMAP)tmpDC.SelectObject(m_bmpNew);
    CBitmap tmpBmp;
    tmpBmp.CreateCompatibleBitmap(&dc, m_bmInfo.bmWidth, m_bmInfo.bmHeight);
    m_memDC->SelectObject(tmpBmp);
    m_memDC->BitBlt(0, 0, m_bmInfo.bmWidth, m_bmInfo.bmHeight, &tmpDC, 0, 0, SRCCOPY);
    tmpDC.SelectObject(m_bmpOld);
}
else
{
    // go through all the pad draw_area areas and find the min and max size...
    int maxx = 0, maxy = 0;
    int nSize = m_padPage->m_drawAreas->GetSize();
    ASSERT(nSize);
    for (int i = 0; i < nSize; i++)
    {
        PadDrawArea &pArea = m_padPage->m_drawAreas->ElementAt(i);
        int x = pArea.m_topLeft.x + pArea.m_size.cx;
        int y = pArea.m_topLeft.y + pArea.m_size.cy;
        if (maxx < x)
            maxx = x;
        if (maxy < y)
            maxy = y;
    }
    // add a little extra space
    maxx += 5;
    maxy += 5;
    m_sizeAll.cx = (maxx > m_sizeClient.cx) ? maxx : m_sizeClient.cx;
    m_sizeAll.cy = (maxy > m_sizeClient.cy) ? maxy : m_sizeClient.cy;
    m_memDC = new CDC();
    CClientDC dc(this);
    m_memDC->CreateCompatibleDC(&dc);
    m_bmpNew.CreateCompatibleBitmap(m_memDC, maxx, maxy);
    m_bmpNew.GetBitmap(&m_bmInfo);
```

```
      m_bmpOld = (HBITMAP)m_memDC->SelectObject(m_bmpNew);
      // now draw areas onto screen
      CPen pen;
      pen.CreatePen(PS_SOLID, 2, RGB(0, 0, 255));
      HPEN oldPen = (HPEN)m_memDC->SelectObject(pen);
      HBRUSH oldBrush = (HBRUSH)m_memDC->SelectStockObject(WHITE_BRUSH);
      CRect rect(0, 0, m_sizeAll.cx, m_sizeAll.cy);
      rect.DeflateRect(1, 1, 1, 1);
      m_memDC->Rectangle(rect);
      m_memDC->SelectStockObject(NULL_BRUSH);
      for (i = 0; i < nSize; i++)
      {
         PadDrawArea &pArea = m_padPage->m_drawAreas->ElementAt(i);
         m_memDC->Rectangle(pArea.m_topLeft.x, pArea.m_topLeft.y,  pArea.m_topLeft.x + pArea.m_size.cx,
pArea.m_topLeft.y + pArea.m_size.cy);
         if (pArea.m_userAreas)
         {
            int uaSize = pArea.m_userAreas->GetSize();
            for (int j = 0; j < uaSize; j++)
            {
               PadArea &uarea = pArea.m_userAreas->ElementAt(j);
               m_memDC->Rectangle(uarea.m_topLeft.x, uarea.m_topLeft.y, uarea.m_topLeft.x + uarea.m_size.cx,
uarea.m_topLeft.y + uarea.m_size.cy);
            }
         }
      }
      m_memDC->SelectObject(oldPen);
      m_memDC->SelectObject(oldBrush);
   }
   double pelsPerUnitx = (.333 * theDoc->m_pelsPerMM_x);
   double pelsPerUnity = (.333 * theDoc->m_pelsPerMM_y);
   int nstrokes = m_strokes.GetSize();
   // here is where the pen strokes are drawn to the memory device context
   CPen pen;
   pen.CreatePen(PS_SOLID, 2, RGB(0, 0, 255));
   HPEN oldPen = (HPEN)m_memDC->SelectObject(pen);
   const double offset = PAGEOFFSET/2.;
   for (int i = 0; i < nstrokes; i++)
   {
      PenStroke &stroke = m_strokes.ElementAt(i);
      int numPoints = stroke.numPoints;
      POINT m_line[2];   // Point array for drawing
      double x = (((double)stroke.points[0].x) / CALLIG_FIXUP + offset) * pelsPerUnitx;
      double y = (((double)stroke.points[0].y) / CALLIG_FIXUP + offset) * pelsPerUnitx;
      m_line[0].x = (int)x;
      m_line[0].y = (int)y;
      for (int j = 0; j < numPoints; j ++ )
      {
         x = ((((double)stroke.points[j].x) / CALLIG_FIXUP) + offset) * pelsPerUnitx;
         y = ((((double)stroke.points[j].y) / CALLIG_FIXUP) + offset) * pelsPerUnitx;
```

```
            m_line[1].x = (int)x;
            m_line[1].y = (int)y;
            if ( HWRAbs(m_line[0].x - m_line[1].x) + HWRAbs(m_line[0].y - m_line[1].y) >= 2)
            {
               m_memDC->Polyline(m_line, 2 );
               m_line[0] = m_line[1];
            }
         }
      }
      m_memDC->SelectObject(oldPen);
      // . . . then it goes on to set up scroll bars...
      return !(m_bRecreate = FALSE);
}
// ********************************************************************************
// 3)  Sending data to text recognition engine:
//
// Following is the main loop for handling incoming pen data:
void AnotoPointToCPoint(FPOINT &anotoPt, CPoint &cgrPoint, CFusionWareEditorDoc *pDoc)
{
      // convert to calligrapher points
      // samples are in Anoto Corrdinates.
      // Convert to short in a resolution that Calligrapher can handle
      double x = anotoPt.X - PAGEOFFSET;
      double y = anotoPt.Y - PAGEOFFSET;
      cgrPoint.x = (int)( x * CALLIG_FIXUP);
      cgrPoint.y = (int)( y * CALLIG_FIXUP);
}
//  DoRecognize
//      This routine is called after all strokes for a given session have been sent
//      to the recognition engine.  It now asks the recognition engine to provide
//      the words, alternates, weights and actual strokes used for each word
BOOL DoRecognize(Calligrapher *m_callig, p_RecogWordData* data, UINT &num_ans, CFusionWareEditorDoc
*pDoc )
{
      TRACE0("DoRecognize called\r\n");
      if (!m_callig->IsLoaded())
         return FALSE;
      UINT   num_alts, num_strokes, weight;
      TCHAR * ptr;
      TCHAR str[256];
#ifdef _DEBUG
      CString strRecognized = _T("");
#endif
      // get the number of words found/recognized
      num_ans = m_callig->GetAnswers( CGA_NUM_ANSWERS, 0, 0 );
      if (num_ans == 0)
         return FALSE;
      (*data) = new RecogWordData[num_ans];
      TRACE1("Answers found: %d\r\n", num_ans);
      for (UINT k = 0; k < num_ans; k++ )
```

```
   {
      // get number of alternates
      num_alts    = m_callig->GetAnswers(CGA_NUM_ALTS, k, 0);
      (*data)[k].numAlts = num_alts;
      // allocate space to store alternatives
      (*data)[k].alts = new LPTSTR[num_alts];
      // allocate place to store weights (percent probability) for each alternate
      (*data)[k].weights = new DWORD[num_alts];
      // get number of strokes used
      num_strokes = m_callig->GetAnswers(CGA_ALT_NSTR, k, 0);
      (*data)[k].numStrokes = num_strokes;
      // get the actual strokes used (may or may not match those sent in
      (*data)[k].strokesUsed = (int *)m_callig->GetAnswers(CGA_ALT_STROKES, k, 0);
      UINT highWeight = 0;
      // now ask for each alternative:  the first one is the one we default to
      // using
      for (UINT i = 0, n = 0; i < num_alts; i++ )
      {
         // get the alternate
         ptr = (TCHAR *)(m_callig->GetAnswers(CGA_ALT_WORD, k, i)); // Get word alternative
         int len = _tcslen(ptr) * sizeof(TCHAR);
         (*data)[k].alts[i] = new TCHAR[len];
         _tcscpy((*data)[k].alts[i], ptr);
         // get this alternates percent probability
         weight = (int)m_callig->GetAnswers(CGA_ALT_WEIGHT, k, i);     // Get weight of the alternative
         (*data)[k].weights[i] = weight;
         if (weight > highWeight)
         {
            highWeight = weight;
            (*data)[k].selIndex = i;
         }
         // DEBUG ONLY: Place answers in the ans buffer
#ifdef _DEBUG
         strRecognized += ptr;
         strRecognized += _T(" : ");
         wsprintf(str, _T("%d"), weight);
         strRecognized +=  str;
         strRecognized += ", ";
#endif
      }
#ifdef _DEBUG
      strRecognized += _T("\r\n>>>");
      wsprintf(str, TEXT("Nstrokes: %2d"), num_strokes);
      strRecognized += _T("\r\n>>>");
      strRecognized += str;
      strRecognized += _T("\r\n>>>");
#endif
   }
#ifdef _DEBUG
   // TRACE will crash is StrRecognized is too long.
```

```
    CString traceOut;
    if (strRecognized.GetLength() > 20)
        traceOut = strRecognized.Left(20);
    else
        traceOut = strRecognized;
    TRACE1("%s\r\n", traceOut);
    pDoc->m_szTestData += strRecognized;
#endif
    return TRUE;
}
Ret_t ProcessPages(InstanceID_t id, VoidPtr_t pUserData, PagesPtr_t pContent)
{
    TRACE0("Enter ProcessPages...\r\n");
    // create test padfile info
    CFusionWareEditorDoc *pDoc = (CFusionWareEditorDoc*)pUserData;
    CStrokesDB &sDB = pDoc->GetStrokesDatabase();
    CProgressCtrl &progCtrl = theMainView->m_progressCtrl;
    theMainView->m_progText.SetWindowText(_T("Processing new pen data . . ."));
    progCtrl.ShowWindow(SW_SHOW);
    Idle();
    pDoc->m_szTestData = _T("");
    //  First get the pen data as pages from the WBXML parser
    CString strText;
    // Now parse the STF Data along with Stylo data
    PageListPtr_t pagelist = NULL;
    PagePtr_t page = NULL;
    pagelist = pContent->pages;
    StfDecoder1_0 stfdecoder;
    float offsetX = 0, offsetY = 0;
    int  iPages = 0;
    while(pagelist)
    {
        // get next in the list
        pagelist = pagelist->next;
        iPages++;
    }
    CPage* pcPage = new CPage[iPages];
    pagelist = pContent->pages;
    iPages = 0;
    while(pagelist)
    {
        // get each page stuff
        page = pagelist->item;
        if( page )
        {
            STFPtr_t stf  = NULL;
            StyloPtr_t stylo = NULL;
            stf = page->stf;
            stylo = page->stylo;
            PcdataPtr_t stylodata = NULL;
```

```
        PcdataPtr_t stfdata = NULL;
        if( stylo )
        {
            stylodata = stylo->table->data;
        }
        if(stf)
        {
            stfdata = stf->data;
        }
        pcPage[iPages].SetSize( CSize(page->width, page->heigth) );
        if(page->pa)
        {
            // To Do make char* to TCHAR*
            wchar_t* lpwszID = NULL;
            int   iLength = 0;
            char* pszID = smlPcdata2String(page->pa);
            LocaleStringToUnicode( pszID, &lpwszID, &iLength );
            pcPage[iPages].Id(lpwszID);
            delete [] lpwszID;
        }
        else
        {
            pcPage[iPages].Id(_T("0.0.0.0"));
        }
        BinaryReader reader = BinaryReader((BYTE*)stfdata->content, stfdata->length );
        stfdecoder.decode( reader, stylodata ? (BYTE*)stylodata->content : NULL, stylodata ? stylodata->length : 0,
pcPage[iPages], offsetX, offsetY );
        iPages++;
      }
      // get next in the list
      pagelist = pagelist->next;
   }
   if (g_padInfo == NULL)
      g_padInfo = new CPadInfo;
   //  Now itererate through the pages and pull out the items
   for (int pageIndex = 0; pageIndex < iPages; pageIndex++)
   {
      if (g_padInfo->SelectPage(pcPage[pageIndex].Id()))
      {
         // page already exists...
         g_padInfo->ResetCurPageData();
      }
      else
      {
         if (!g_padInfo->LoadPadFile((LPCTSTR)g_padFileName, pcPage[pageIndex].Id()))
            continue;  // try next page
         g_padInfo->SelectPage(pcPage[pageIndex].Id());
      }
      CString strFormat;
      CString strData;
```

```
    CStroke* pStroke = NULL;
    vector<CStroke>& strokes = pcPage[pageIndex].GetStrokes( );
    UINT numStrokes = strokes.size();
    double tLastTime = 0.;
    g_padInfo->SetCurPageCreateTime(strokes[0].StartSecond());
    progCtrl.SetRange(0, numStrokes * 2 + g_padInfo->m_pages.GetSize() + 2);
    progCtrl.SetStep(1);
    // setup calligrapher
    PenStrokes pAllStrokes = new PenStroke[numStrokes];
    pDoc->m_callig->CleanStorage();
    // start new section
    TRACE0("StartSect called...\r\n");
    pDoc->m_callig->StartSect();
    UINT wordsFound;          // number of words found
    // store off first time
    // Main loop for converting Anoto floating point strokes to
    // integer values and storing them and other relevant data
    // in the DB
    if (numStrokes)
        tLastTime = strokes[0].Time();
#ifdef _DEBUG
    short minX, minY, maxX, maxY;
    minX = minY = SHRT_MAX;
    maxX = maxY = 0;
#endif
    for( UINT i = 0; i < numStrokes; i++ )
    {
        pStroke = &strokes[i];
        vector<FPOINT>& samples = pStroke->Samples();
        CPoint *cPoints = new CPoint[samples.size()];
        pDoc->m_callig->StartStroke();
        UINT numSamples = samples.size();
        double tTime = pStroke->Time();
        // if time difference between strokes is greater than 1200ms
        // start new section.
        if ((tTime - tLastTime) > 1200)
        {
            TRACE0("closing callig section due to time difference\r\n");
            pDoc->m_callig->EndSect();
            TRACE0("EndSect called...\r\n");
            // start another since we have another stroke at least
            pDoc->m_callig->StartSect();
            TRACE0("StartSect called...\r\n");
        }
        tLastTime = tTime + pStroke->Duration();
        TRACE1("Adding %d points\r\n", numSamples);
        // the Anoto points come in as floating point values, which must be translated to integer for the recognizer
        for (UINT j = 0; j < numSamples; j++)
        {
            // convert from float to integer
```

```
            AnotoPointToCPoint(samples[j], cPoints[j], pDoc);
            // add to recognizer
#ifdef _DEBUG
            // find the bounds for debug purposes
            int x, y;
            x = cPoints[j].x;
            y = cPoints[j].y;
            if (x < minX)
               minX = x;
            else if (x > maxX)
               maxX = x;
            if (y < minY)
               minY = y;
            else if (y > maxY)
               maxY = y;
#endif
         }
         // save points in temp array
         pAllStrokes[i].points = cPoints;
         pAllStrokes[i].numPoints = numSamples;
         progCtrl.StepIt();
         Idle();
         // now find section for this item
         pDoc->m_callig->EndStroke();
      }
      pDoc->m_callig->EndSect();
      TRACE0("EndSect called...\r\n");
      // The points have been converted, now send them
      // to the recognizer, and fill the RecogData structure,
      // Each RecogData structure will contain the strokes, words and alternate words found
      //  and each is associated with a Page location or PadArea
      CMap<PadArea*, PadArea*&, PadArea*, PadArea*&> padMap;
      PadArea *lastArea = NULL;
      // divide strokes up into areas as defined by PAD file
      for (i = 0; i < numStrokes; i++)
      {
         //  see below for implementation of FindPadArea
         PadArea *area = g_padInfo->FindPadArea(pAllStrokes[i].points, pAllStrokes[i].numPoints);
         if (!area)
         {
            TRACE0("Received stroke that was not associated with any page location!");
            if (lastArea)
               lastArea->m_strokes->Add(pAllStrokes[i]);
            continue;
         }
         TRACE1("Stroke added to area %s\r\n", (LPCTSTR)area->m_name);
         //  keep pad areas in a map for easy look each iteration through this loop
         if (!padMap.Lookup(area, area))
         {
            padMap[area] = area;
```

```
#ifdef ZFORTIFY
#undef new
#endif
          area->m_strokes = new CArray<PenStroke, PenStroke&>;
#ifdef ZFORTIFY
  #define new ZFortify_New
#endif
       }
       ASSERT(area->m_strokes);
       area->m_strokes->Add(pAllStrokes[i]);
       lastArea = area;
       progCtrl.StepIt();
       Idle();
    }
    // now send each area's strokes to recog engine
    POSITION pos = padMap.GetStartPosition();
    Calligrapher *pCallig = pDoc->m_callig;
    while (pos != NULL)
    {
       PadArea *area, *area1;
       padMap.GetNextAssoc(pos, area, area1);
       ASSERT(area == area1);
       CArray<PenStroke, PenStroke&> *strokes = area->m_strokes;
       int nStrokes = strokes->GetSize();
#ifdef ZFORTIFY
#undef new
#endif
       if (!area->m_wordData)
          area->m_wordData = new CArray<RecogWordData, RecogWordData&>;
#ifdef ZFORTIFY
  #define new ZFortify_New
#endif
       if (nStrokes)
       {
          // recognition engine called here for each page/PAD section
          pCallig->OpenDefaultSession();
          for (i = 0; i < (UINT)nStrokes; i++)
          {
             PenStroke &stroke = (*strokes)[i];
             pCallig->Recognize(stroke.numPoints, stroke.points ); // And send it to recognition
          }
          pCallig->Recognize(0, NULL); // end section
          pCallig->CloseSession();
          p_RecogWordData recogData = NULL;  // array of recognized words
          if (DoRecognize(pDoc->m_callig, &recogData, wordsFound, pDoc))
          {
             TRACE1("DoRecog found %d words\r\n", wordsFound);
             FillRecogWordData(recogData, wordsFound, strokes, pDoc);
             for (UINT k = 0; k < wordsFound; k++)
             {
```

```
                    //                  recogData[k].id = pDoc->GenerateUID();
                    recogData[k].pageID = 1;
                    //                  recogData[k].StoreData(sDB);
                    recogData[k].padArea = area;
                    recogData[k].bDeleteData = FALSE;
                    area->m_wordData->Add(recogData[k]);
                  }
                  delete[] recogData;
                }
              }
              progCtrl.StepIt();
            }
            delete[] pAllStrokes;
            // g_padInfo now has all the words
            TRACE2("minX = %d, minY = %d\r\n", minX, minY);
            TRACE2("maxX = %d, maxY = %d\r\n", maxX, maxY);
        }  // for (pageIndex = ...
        pDoc->UpdateAllViews(NULL);
        sDB.Close();
        TRACE0("...Exit ProcessPages\r\n");
        progCtrl.ShowWindow(SW_HIDE);
        theMainView->m_progText.SetWindowText(_T("Waiting for new pen data . . ."));
        return SML_ERR_OK;
}
//  Following is the code that is used to determine which page/PAD area a give stroke belongs in
//  it essentially gets the bounds of the stroke and determines which page/PAD area rectangle contains the majority
//  of the stroke bounds  There are various ways to optimize this, but the current implementation is straightforward
//  The result of this code is that via the page definition files, we know what type of item the strokes create - an event
//  task, note, email, contact, etc.
// These first two functions are used to determine if an area has ANY of the bounding vertices
// returns number of vertices of testRect in checkRect, if any
int NumRectPointsInRect(CRect &checkRect, CRect &testRect)
{
    int count = 0;
    if (checkRect.PtInRect(testRect.TopLeft()))
        count++;
    if (checkRect.PtInRect(testRect.BottomRight()))
        count++;
    CPoint ptTopRight(testRect.right, testRect.top);
    if (checkRect.PtInRect(ptTopRight))
        count++;
    CPoint ptBottomLeft(testRect.left, testRect.bottom);
    if (checkRect.PtInRect(ptBottomLeft))
        count++;
    return count;
}
BOOL PadArea::ContainsStroke(CRect &bounds)
{
    int count = NumRectPointsInRect(this->GetRect(), bounds);
    //  if it has more than two vertices, it MUST contain all of the stroke rectangle
```

```
    if (count > 2)
        return TRUE;
    // check to see if this area contain MOST of the bounds rect.
    CRect intRect;
    intRect.IntersectRect(this->GetRect(), bounds);
    DWORD barea = bounds.Height() * bounds.Width();
    DWORD iarea = intRect.Width() * intRect.Height();
    //    TRACE2("StrokeArea = %ld, IntersectArea = %ld\r\n", barea, iarea);
    if ((barea/2) < iarea)
        return TRUE;
    return FALSE;
}
// FindPadARea gets the bounds of the stroke and iterates the PAD areas to see which, if any, contain MOST of the
stroke area
//
PadArea *CPadInfo::FindPadArea(CPoint *stroke, UINT numPoints)
{
    ASSERT(numPoints);
    // get the bounds of the stroke
    CRect bounds(stroke[0].x, stroke[0].y, stroke[0].x, stroke[0].y);
    long &xMin = bounds.left;
    long &yMin = bounds.top;
    long &xMax = bounds.right;
    long &yMax = bounds.bottom;
    for (UINT i = 1; i < numPoints; i++)
    {
        if (xMin > stroke[i].x)
            xMin = stroke[i].x;
        else if (xMax < stroke[i].x)
            xMax = stroke[i].x;
        if (yMin > stroke[i].y)
            yMin = stroke[i].y;
        else if (yMax < stroke[i].y)
            yMax = stroke[i].y;
    }
    double pelsPerUnitx = (.333 * theDoc->m_pelsPerMM_x);
    // now adjust to PAD points
    xMin = PAD_ADJUST(xMin)*pelsPerUnitx;
    yMin = PAD_ADJUST(yMin)*pelsPerUnitx;
    xMax = PAD_ADJUST(xMax)*pelsPerUnitx;
    yMax = PAD_ADJUST(yMax)*pelsPerUnitx;
    // got bounding rect, see what area it is in.
    PadPage &pPage = this->m_pages[m_pageIndex];
    ASSERT(pPage.m_drawAreas);
    i = 0;
    BOOL done = FALSE;
    PadDrawArea *pdrawArea, *foundDrawArea = NULL;
    CArray<PadDrawArea, PadDrawArea&> *drawAreas = pPage.m_drawAreas;
    for (i = 0; i < (UINT)drawAreas->GetSize(); i++)
    {
```

```
        pdrawArea = &drawAreas->ElementAt(i);
        if (pdrawArea->ContainsStroke(bounds))
        {
            foundDrawArea = pdrawArea;
            break;
        }
    }
    if (foundDrawArea && foundDrawArea->m_userAreas)
    {
        PadArea *pArea, *foundArea = NULL;
        UINT size = foundDrawArea->m_userAreas->GetSize();
        for (i = 0; i < (UINT)size; i++)
        {
            pArea = &foundDrawArea->m_userAreas->ElementAt(i);
            if (pArea->ContainsStroke(bounds))
            {
                foundArea = pArea;
                break;
            }
        }
        if (!foundArea)
        {
            TRACE0("*** Draw area found, but no user area found!!!");
            return (PadArea*)foundDrawArea;
        }
        TRACE1("Area found name: %s", foundArea->m_name);
        return foundArea;
    }
    else
    {
        TRACE0("*** Draw area found, but no user area found!!!");
        return (PadArea *)foundDrawArea;
    }
}
// ****************************************************************************
// 4)  Allowing the user to select a list of alternates
// This is one of the more complicated parts of the code. When the user click
// and holds on an edit control, a context menu is created, which may be
// populated with alternate words if they exist.
void CEditCtrl::ContextMenu(CPoint point)
{
    CMenu mnuCtxt;
    if (!m_nMenuID)
        return;
    mnuCtxt.CreatePopupMenu();
    // . . .
    // now see if there are alternate words for word at this location
    int x = point.x - GetMarginWidth();
    int y = point.y;
    CPoint curLoc;
```

```
   MouseToCaret(x, y, curLoc);
   m_nContextMenuIndex = -1;
   if (curLoc.y < m_vLines.GetSize())
   {
      Line &lLine = m_vLines[curLoc.y];
      // The wordAlternates structure contains the info about alternate words for a given word starting at a given line
index
      if (lLine.wordAlternates)
      {
         m_ptCaretPos = curLoc;
         CPoint start, end;
         // from the given word point location, we get the word bounds
         GetWordPoints(start, end);
         WordAltInfo *p_wordInfo = NULL;
         BOOL bFound = FALSE;
         // then see if an alternate exists for the given word index
         for (int i = 0; i < lLine.wordAlternates->GetSize(); i++)
         {
            p_wordInfo = &(lLine.wordAlternates->ElementAt(i));
            int index = p_wordInfo->index;
            if (index >= start.x && index <= end.x)
            {
               m_nContextMenuIndex = i;
               break;
            }
         }
         if (m_nContextMenuIndex != -1 && p_wordInfo && p_wordInfo->wordData && (p_wordInfo->wordData-
>numAlts > 1))
         {
            if (menuAdded)
               mnuCtxt.AppendMenu(MF_SEPARATOR, 0, _T(""));
            RecogWordData *wordData = p_wordInfo->wordData;
            for (i = 0; i < wordData->numAlts  && i <= EM_INSERTALTLAST; i++)
            {
               if (i != wordData->selIndex)
               {
                  mnuCtxt.AppendMenu(MF_ENABLED, EM_INSERTALT+i, wordData->alts[i]);
                  menuAdded++;
               }
            }
         }
         MakeCaretVisible();
      }
   }
   if (menuAdded)
      mnuCtxt.TrackPopupMenu(TPM_LEFTALIGN|TPM_TOPALIGN, point.x, point.y, this);
}
// The complicated part is keeping the wordInfo array up to date if the user
// does any editing of the items.  We have to update the wordInfo items if
// the user cuts, pastes, deletes, backspaces, undo, redo, etc, etc so that
```

```
// the indexes stay associated with the correct words, even if said words have
// been modified by the user.  We only drop the wordInfo item if the first
// letter of the word is deleted.  But if the user does an undo of that, we
// have to re-add the item.  Following are bits of code that do most of this:
// NOTE: this code is only unit tested and possibly incomplete at this point.
// Usually, alternate word info is created when a new line is inserted,
// if the code doing the insertion passes in an alt info array
BOOL CEditCtrl::InsertLine(LPCTSTR text, int nLength, int nPosition, WordAltInfoArrayPtr p_wordInfo, int wailndex)
{
   Line line;
   if (nLength == -1)
   {
      if (text != NULL)
         nLength = (int)_tcslen(text);
   }
   line.nLength = nLength;
   line.nMax = (nLength ? ROUND_SIZE(REAL_SIZE(nLength)): 0);
   // line.nBlockColor = -1;
   if (nLength != 0)
   {
      line.pcText = new TCHAR[line.nMax+1];
      if (!line.pcText)
         return FALSE;
      if (p_wordInfo)
      {
         line.wordAlternates = new WordAltInfoArray;
         for (int i = 0; i < p_wordInfo->GetSize(); i++)
         {
            if (wailndex)
            {
               WordAltInfo item = p_wordInfo->ElementAt(i);
               int lastIndex = wailndex + nLength;
               if (item.index >= wailndex && item.index <= lastIndex)
               {
                  item.index -= wailndex;
                  line.wordAlternates->Add(item);
               }
            }
            else
            {
               WordAltInfo item = p_wordInfo->ElementAt(i);
               if (item.index < nLength)
                  line.wordAlternates->Add(item);
            }
         }
      }
      memset(line.pcText, 0, REAL_SIZE(nLength+1));
      memcpy(line.pcText, (LPCTSTR)text, REAL_SIZE(nLength));
   }
   if (nPosition == -1)
```

```
        m_vLines.Add(line);
    else
    {
        m_vLines.InsertAt(nPosition, line);
    }
    nLength = (int)CaretOffsetLine(nPosition == -1 ? (int)m_vLines.GetSize() - 1 : nPosition, nLength);
    if (nLength > m_nLongestLine)
    {
        m_nLongestLine = nLength;
        SetupHScroller();
    }
    return TRUE;
}
BOOL CEditCtrl::InsertTextEx(int nLine, int nColumn, LPCTSTR lpszText, CPoint *ptNewPos, WordAltInfoArrayPtr
p_wordInfo )
{
    TCHAR  *pcRemChars = NULL;
    int nRemaining = 0, nCurrent = nLine, nIndex;
    BOOL bError = FALSE;
    WordAltInfoArrayPtr tmpWordInfo = NULL;
    // First we save the characters right
    // of the caret on the line we start
    // the insert on.
    if (m_vLines.GetSize() > nLine)
    {
        Line &lLine = m_vLines[nLine];
        tmpWordInfo = lLine.wordAlternates;
        // Any text?
        // . . .
        int cumulative = 0;
        while ( 1 )
        {
            // Reset index.
            nIndex = 0;
            // Iterate the text until we reach a line
            // break or the end of the text.
            int width = nColumn;
            while ( lpszText[ nIndex ] != _T( '\0' )
                    && lpszText[ nIndex ] != _T( '\r' )
                    && lpszText[ nIndex ] != _T( '\n' )
                  )
                nIndex++;
            // Are we still on the current line?
            if ( nCurrent == nLine )
            {
                // Yes. Append the characters.
                bError = !AppendText(lpszText, nCurrent, nIndex, p_wordInfo );
                // not a new line, but has alternate word info?
                if (!bError && tmpWordInfo)
                {
```

```
            int size = tmpWordInfo->GetSize();
            for (int i = 0; i < size; i++)
            {
               // add in new characters
               if (tmpWordInfo->ElementAt(i).index > nColumn)
                  tmpWordInfo->ElementAt(i).index += nIndex;
               TRACE1("INSERRTTEXTEX::WordInfo index: %d\r\n", tmpWordInfo->ElementAt(i).index);
            }
         }
      }
      else
      {
         // Insert a new line.
         bError = ! InsertLine(lpszText, nIndex, nCurrent, p_wordInfo,  cumulative);
      }
      // Did we reach the end of the string?
      if ( lpszText[ nIndex ] == _T( '\0' ))
      {
         // Setup the new position.
         if ( ptNewPos != NULL )
         {
            ASSERT(m_vLines.GetUpperBound() >= nCurrent);
            Line &line = m_vLines[nCurrent];
            ptNewPos->x = line.nLength;
            ptNewPos->y = nCurrent;
         }
         // Any remaining characters to add?
         if ( pcRemChars )
            bError = ! AppendText(pcRemChars, nCurrent, nRemaining );
         break;
      }
      // Next line...
      nCurrent++;
      nIndex++;
      // Newline?
      if ( lpszText[ nIndex ] == _T( '\n' ) || lpszText[ nIndex ] == _T( '\r' ))
         nIndex++;
      cumulative += nIndex;
      // Adjust pointer.
      lpszText += nIndex;
   }
   // . . .
  }
}
BOOL CEditCtrl::AppendText(LPCTSTR lpszText, int nLine, int nLength, WordAltInfoArrayPtr p_wordInfo, int
wordAltIndexStart )
{
  // Length passed?
  // . . .
  // Any text to append?
```

```
   // . . .
   // checks here to see if a newly inserted item is a substitute word due to the user selecting an alternate from the
Context Menu
   if (p_wordInfo)
   {
      // inserting substitute word?
      if (lLine.wordAlternates)
      {
         // no need to adjust anything here
         // just add the new wordInfo stuff
         int size = p_wordInfo->GetSize();
         for (int i = 0; i < size; i++)
            lLine.wordAlternates->Add(p_wordInfo->ElementAt(i));
      }
      else // no, this is new
      {
         // adjust index's per nLength
         int size  = p_wordInfo->GetSize();
         if (lLine.nLength)
         {
            for (int i = 0; i < size; i++)
            {
               p_wordInfo->ElementAt(i).index += lLine.nLength;
               TRACE1("APPENDTEXT::wordInfo index = %d\r\n",  p_wordInfo->ElementAt(i).index);
            }
         }
         lLine.wordAlternates = new WordAltInfoArray;
         // is this a previously existing line?
         if (wordAltIndexStart)
         {
            int len = lLine.nLength;
            for (int i = 0; i < size; i++)
            {
               // see if this has any words with alternates
               // this would be alts with index's greater than wordAltIndexStart
               // include len that was added on above
               if (p_wordInfo->ElementAt(i).index > (wordAltIndexStart + len))
                  lLine.wordAlternates->Add(p_wordInfo->ElementAt(i));
            }
         }
         else    // these seperated for speed's sake, even though possible to combine lines above with this.
            for (int i = 0; i < size; i++)
               lLine.wordAlternates->Add(p_wordInfo->ElementAt(i));
      }
   }
   // . . .
}
// There is similar code for deleting text, cutting text, inserting text, Undo and
// Redo.  This code essentially tracks the WordAlternate
// indexes and adjusts them, adds them, deletes them as necessary.
```

```
// ********************************************************************************
// 5) Constructing an event from the data
BOOL CSelectItemsDlg::OnInitDialog()
{
   // . . .
   SYSTEMTIME sysTime;
   CString  strText;
   COleDateTime dt = COleDateTime::GetCurrentTime();
   dt.GetAsSystemTime( sysTime );
   The strokes and word data have already been sorted into Page/PAD areas which themselves constitute an event,
task, etc.  This code goes through and finds each area by type.
   // Is there any "real" data?
   if (g_padInfo && g_padInfo->m_pages.GetSize() > 0)
   {
      UINT nSize = g_padInfo->m_pages.GetSize();
      for (UINT i = 0; i < nSize; i++)
      {
         PadPage *page = &g_padInfo->m_pages.ElementAt(i);
         ASSERT(page);
         UINT pdSize = page->m_drawAreas->GetSize();
         for (UINT j=0; j < pdSize; j++)
         {
            PadDrawArea &pdArea = page->m_drawAreas->ElementAt(j);
            if (pdArea.m_userAreas)
            {
               UINT uaSize = pdArea.m_userAreas->GetSize();
               for (UINT k = 0; k < uaSize; k++)
               {
                  PadArea &uaArea = pdArea.m_userAreas->ElementAt(k);
                  if ((uaArea.m_strokes != NULL) && (uaArea.m_strokes->GetSize()))
                  {
                     CString aName = uaArea.m_name;
                     if (aName.Left(11).CompareNoCase(L"appointment") == 0)
                     {
                        CFusionEvent* pEvent = new CFusionEvent;
                        CTime curDate(sysTime);
                        if (page->m_appInfoAreas)
                        {
                           int aiSize = page->m_appInfoAreas->GetSize();
                           PadArea *aiArea;
                           BOOL bFound = FALSE;
                           for (int ai = 0; ai < aiSize; ai++)
                           {
                              aiArea = &(page->m_appInfoAreas->ElementAt(ai));
                              if (aiArea->m_name == L"date")
                              {
                                 bFound = TRUE;
                                 break;
                              }
                           }
```

```
        if (bFound)
        {
            curDate = StrToCTime(aiArea->m_value);
        }
    }
    // time should be the last char or two at the end of the area name.
    int curTime = _wtoi((LPCTSTR)aName.Right(2));
    curDate = CTime(curDate.GetYear(), curDate.GetMonth(), curDate.GetDay(), curTime, 0, 0);
    strText = uaArea.ExpandStringData();
    pEvent->SetSubject(strText);
    TRACE1("Subject: %s\r\n", (LPCTSTR)strText);
    pEvent->SetIcon( eNewAppoint );
    SYSTEMTIME tTime;
    curDate.GetAsSystemTime(tTime);
    pEvent->SetStartDT( tTime );
    curDate = CTime(curDate.GetYear(), curDate.GetMonth(), curDate.GetDay(), curTime+1, 0, 0);
    curDate.GetAsSystemTime(tTime);
    pEvent->SetEndDT( tTime );
    pEvent->SetRecordDT( sysTime );
    pEvent->m_padPageIndex = i;
    pEvent->m_padArea = &uaArea;
    __int64 createTime = page->m_createTime;
    ULARGE_INTEGER uli = *(ULARGE_INTEGER *)&createTime;
    FILETIME ft = *(FILETIME *)&uli;
    SYSTEMTIME sysTime;
    FileTimeToSystemTime(&ft, &sysTime);
    pEvent->SetRecordDT(sysTime);
    m_ItemsList.AddItem( pEvent );
    // we found an appointment.
}
else if (aName.Left(4).CompareNoCase(L"task") == 0)
{
    // we found a task
    CFusionTask *pTask = new CFusionTask;
    strText = uaArea.ExpandStringData();
    pTask->SetSubject(strText);
    TRACE1("Subject: %s\r\n", (LPCTSTR)strText);
    pTask->SetIcon( eTask );
    pTask->SetRecordDT( sysTime );
    pTask->m_padArea = &uaArea;
    pTask->m_padPageIndex = i;
    __int64 createTime = page->m_createTime;
    ULARGE_INTEGER uli = *(ULARGE_INTEGER *)&createTime;
    FILETIME ft = *(FILETIME *)&uli;
    SYSTEMTIME sysTime;
    FileTimeToSystemTime(&ft, &sysTime);
    pTask->SetRecordDT(sysTime);
    CTime curDate(sysTime);
    if (page->m_appInfoAreas)
    {
```

```
                        int aiSize = page->m_appInfoAreas->GetSize();
                        PadArea *aiArea;
                        BOOL bFound = FALSE;
                        for (int ai = 0; ai < aiSize; ai++)
                        {
                            aiArea = &(page->m_appInfoAreas->ElementAt(ai));
                            if (aiArea->m_name == L"date")
                            {
                                bFound = TRUE;
                                break;
                            }
                        }
                        if (bFound)
                        {
                            curDate = StrToCTime(aiArea->m_value);
                        }
                    }
                    SYSTEMTIME tTime;
                    curDate.GetAsSystemTime(tTime);
                    //      pTask->SetStartDT( tTime );
                    m_ItemsList.AddItem( pTask );
                }
                else if (aName.Left(4).CompareNoCase(L"note") == 0)
                {
                    // we found a note
                    CFusionNote *pNote = new CFusionNote;
                    strText = uaArea.ExpandStringData();
                    pNote->SetSubject(strText);
                    TRACE1("Subject: %s\r\n", (LPCTSTR)strText);
                    pNote->SetIcon( ePlainNote );
                    pNote->SetRecordDT( sysTime );
                    pNote->m_padPageIndex = i;
                    pNote->m_padArea = &uaArea;
                    __int64 createTime = page->m_createTime;
                    ULARGE_INTEGER uli = *(ULARGE_INTEGER *)&createTime;
                    FILETIME ft = *(FILETIME *)&uli;
                    SYSTEMTIME sysTime;
                    FileTimeToSystemTime(&ft, &sysTime);
                    pNote->SetRecordDT(sysTime);
                    m_ItemsList.AddItem( pNote );
                }
            }
        }
    }
}
// ********************************************************************************
// 6) Sending resultant event to database
```

```
//
// In this version, that means sending it to Pocket Outlook.  Each type class, such
// as CFusionEvent and CFusionTask, etc, has a WriteToOutlook function, which is
// simply called when the user clicks on a SendToOutlook button, menu, etc.
// The code is as follows in general, and is pretty much as required by the
// Pocket Outlook Object Model interface:
bool CFusionEvent::WriteToOutlook( void* pOutlookManager )
{
    bool   bResult   = true;
    CFWPOOMManager* pManager  = (CFWPOOMManager*)pOutlookManager;
    IAppointment *pAppointment = NULL;
    HRESULT   hr    = S_OK;
    if( !pManager->Initialized() )
        return false;
    // Create an event to receive the new data
    hr = pManager->GetApplication( )->CreateItem( olAppointmentItem, (IDispatch **)&pAppointment );
    // subject
    pAppointment->put_Subject((LPTSTR)(LPCTSTR)m_strSubject);
    // body/description/notes
    pAppointment->put_Body((LPTSTR)(LPCTSTR)m_strBody);
    // location
    pAppointment->put_Location((LPTSTR)(LPCTSTR)m_strLocation);
    // set all day
    // LOOK: inputs are assumed in local, if not change this
    bool bLocalTime = false;
    if (m_bAllday)
    {
        pAppointment->put_AllDayEvent(VARIANT_TRUE);
        //  bLocalTime = false;
    }
    else
    {
        pAppointment->put_AllDayEvent(VARIANT_FALSE);
    }
    // start date
    DATE date = SystemTimeToDATE( m_startDateTime, bLocalTime, pManager );
    pAppointment->put_Start(date);
    // end date
    date = SystemTimeToDATE( m_endDateTime, bLocalTime, pManager );
    pAppointment->put_End(date);
    // write to store
    hr = pAppointment->Save( );
    if( FAILED( hr ) )
    {
        goto cleanup;
    }
cleanup:
    if( pAppointment )
        pAppointment->Release();
    if( FAILED( hr ) )
```

```
        bResult = false;
    return bResult;
}
```